unit_test

July 22, 2020

1 Test Your Algorithm

1.1 Instructions

1. From the **Pulse Rate Algorithm** Notebook you can do one of the following:

- Copy over all the **Code** section to the following Code block.
- Download as a Python (`.py`) and copy the code to the following Code block.

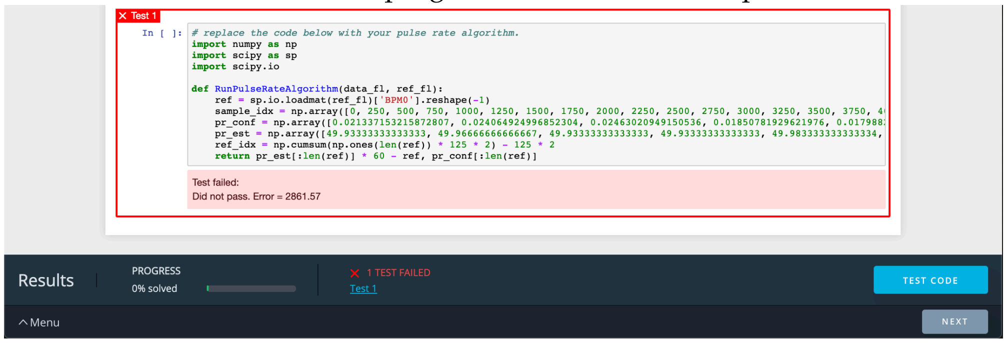
2. In the bottom right, click the Test Run button.

1.1.1 Didn't Pass

If your code didn't pass the test, go back to the previous Concept or to your local setup and continue iterating on your algorithm and try to bring your training error down before testing again.

1.1.2 Pass

If your code passes the test, complete the following! You **must** include a screenshot of your code and the Test being **Passed**. Here is what the starter filler code looks like when the test is run and should be similar. A passed test will include in the notebook a green outline plus a box with **Test passed:** and in the Results bar at the bottom the progress bar will be at 100% plus a checkmark with **All cells passed**.



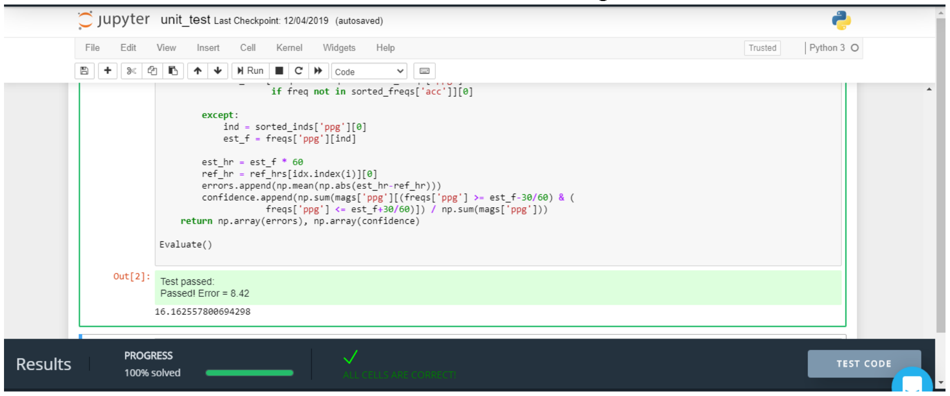
```
# replace the code below with your pulse rate algorithm.
import numpy as np
import scipy as sp
import scipy.io

def RunPulseRateAlgorithm(data_fl, ref_fl):
    ref = sp.io.loadmat(ref_fl)['BPM0'].reshape(-1)
    sample_idx = np.array([0, 250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 2750, 3000, 3250, 3500, 3750, 4
    pr_conf = np.array([0.021337153215872807, 0.024064924996852304, 0.02463020949150536, 0.01850781929621976, 0.017988
    pr_est = np.array([49.93333333333333, 49.96666666666667, 49.93333333333333, 49.93333333333333, 49.983333333333334,
    ref_idx = np.cumsum(np.ones(len(ref)) * 125 * 2) - 125 * 2
    return pr_est[:len(ref)] * 60 - ref, pr_conf[:len(ref)]
```

```
Test failed:
Did not pass. Error = 2861.57
```

1. Take a screenshot of your code passing the test, make sure it is in the format `.png`. If not a `.png` image, you will have to edit the Markdown render the image after Step 3. Here is an example of what the `passed.png` would look like
2. Upload the screenshot to the same folder or directory as this jupyter notebook.

3. Rename the screenshot to `passed.png` and it should show up below.


jupyter unit_test Last Checkpoint: 12/04/2019 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
                if freq not in sorted_freqs['acc']][0]

        except:
            ind = sorted_inds['ppg'][0]
            est_f = freqs['ppg'][ind]

        est_hr = est_f * 60
        ref_hr = ref_hrs[idx.index(i)][0]
        errors.append(np.mean(np.abs(est_hr-ref_hr)))
        confidence.append(np.sum(mags['ppg'][(freqs['ppg'] >= est_f-30/60) & (
                freqs['ppg'] <= est_f+30/60)]) / np.sum(mags['ppg']))
    return np.array(errors), np.array(confidence)

Evaluate()
```

```
Out[2]: Test passed:
        Passed! Error = 8.42
        16.162557800694298
```

Results
PROGRESS
100% solved
ALL CELLS ARE CORRECT!
TEST CODE

4. Download this jupyter notebook as a `.pdf` file.
5. Continue to Part 2 of the Project.

```
In [2]: import glob
        import numpy as np
        import scipy as sp
        import scipy.stats
        import scipy.io
        import scipy.signal
        import matplotlib.pyplot as plt
        import pandas as pd
        from tqdm import tqdm


        def LoadTroikaDataset():
            """
            Retrieve the .mat filenames for the troika dataset.
            Review the README in ./datasets/troika/ to understand the
            organization of the .mat files.

            Returns:
                data_fls: Names of the .mat files that contain signal data
                ref_fls: Names of the .mat files that contain reference data
                <data_fls> and <ref_fls> are ordered correspondingly, so that
                ref_fls[5] is the reference data for data_fls[5], etc...
            """
            data_dir = "./datasets/troika/training_data"
            data_fls = sorted(glob.glob(data_dir + "/DATA_*.mat"))
            ref_fls = sorted(glob.glob(data_dir + "/REF_*.mat"))
            return data_fls, ref_fls


        def LoadTroikaDataFile(data_fl):
            """
            Loads and extracts signals from a troika data file.
```

```
    Usage:
        data_fls, ref_fls = LoadTroikaDataset()
        ppg, accx, accy, accz = LoadTroikaDataFile(data_fls[0])

    Args:
        data_fl: (str) filepath to a troika .mat file.

    Returns:
        numpy arrays for ppg, accx, accy, accz signals.
    """
    data = sp.io.loadmat(data_fl)['sig']
    return data[2:]


def AggregateErrorMetric(pr_errors, confidence_est):
    """
    Computes an aggregate error metric based on confidence estimates.
    Computes the MAE at 90% availability.

    Args:
        pr_errors: a numpy array of errors between pulse rate estimates
        and corresponding reference heart rates.
        confidence_est: a numpy array of confidence estimates
        for each pulse rate error.

    Returns:
        the MAE at 90% availability
    """
    # Higher confidence means a better estimate. The best 90% of the estimates
    #    are above the 10th percentile confidence.
    percentile90_confidence = np.percentile(confidence_est, 10)

    # Find the errors of the best pulse rate estimates
    best_estimates = np.abs(pr_errors[confidence_est >= percentile90_confidence])

    # Return the mean absolute error
    return np.mean(best_estimates)


def Evaluate():
    """
    Top-level function evaluation function.

    Runs the pulse rate algorithm on the Troika dataset and
    returns an aggregate error metric.

    Returns:
        Pulse rate error on the Troika dataset. See AggregateErrorMetric.
    """
    # Retrieve dataset files
    data_fls, ref_fls = LoadTroikaDataset()
    errs, confs = [], []
    for data_fl, ref_fl in zip(data_fls, ref_fls):
        # Run the pulse rate algorithm on each trial in the dataset
        errors, confidence = RunPulseRateAlgorithm(data_fl, ref_fl)
        errs.append(errors)
        confs.append(confidence)
        # Compute aggregate error metric
    errs = np.hstack(errs)
    confs = np.hstack(confs)
    return AggregateErrorMetric(errs, confs)

fs = 125
window_len_s = 10
window_shift_s = 2
past_window = 3
pass_band = (60/60.0, 200/60.0)
multiplier = 4
ppg_mag_height = 0.55
acc_mag_height = 0.3
ppg_min_dist = 0.2
num_best = 2
acc_num_best_arg = 2


def BandpassFilter(signal):
    """
    bandpass_filter
    Loads the signal and passes it through a Butterworth filter.
    Args:
        signal: sinal Data from sensors
    Returns:
        Band Pass filtered Signal
    """
    # Initialising Buterworth Bandpass Filter
    b, a = scipy.signal.butter(3, pass_band, btype='bandpass', fs=fs)
    '''Returns the signal after applying digital butterworth filter
    forward and backward to a signal.'''
    return scipy.signal.filtfilt(b, a, signal)
```

```
def fft(sig, fs):
    freqs = np.fft.rfftfreq(len(sig), 1/fs)
    fft_mag = np.abs(np.fft.rfft(sig))
    return (freqs, fft_mag)

def RunPulseRateAlgorithm(data_fl, ref_fl):
    """
    Args:
        data_fl: (str) filepath to a troika .mat file (signal).
        ref_fl: (str) filepath to a troika .mat file (ground truth heart rate).

    Returns:
        pr_errors: a numpy array of errors between pulse rate estimates and
        corresponding reference heart rates.
        confidence_est: a numpy array of confidence estimates for each pulse
        rate error.
    """
    fs = 125

    # Load ground truth heart rate
    ref_hrs = sp.io.loadmat(ref_fl)['BPM0']

    # Load data using LoadTroikaDataFile
    ppg, accx, accy, accz = LoadTroikaDataFile(data_fl)
    acc = np.mean([accx, accy, accz], axis=0)
    data_list = [ppg, acc]
    label_list = ['ppg', 'acc']

    # Bandpass filter the signal between 70 and 190 BPM
    filtered = {label: BandpassFilter(data) for (label, data) in zip(label_list, data_li

    # Move with a window_length_s of 8s and the window_shift_s of 2s
    # The ground truth data follows the same cadence
    errors, confidence = [], []
    window_length_s = 10
    window_shift_s = 2
    window_length = window_length_s * fs
    window_shift = window_shift_s * fs
    idx = list(range(0, len(ppg) - window_length, window_shift))
    for i in idx:
        segments = {label: filtered[label][
            i: i + window_length] for label in label_list}

        freqs, mags, sorted_inds, sorted_freqs = {}, {}, {}, {}
        for label in label_list:
            freqs[label], mags[label] = fft(segments[label], fs)
            sorted_inds[label] = np.argsort(mags[label])[::-1][:4]
            sorted_freqs[label] = freqs[label][sorted_inds[label]]
            try:
                est_f = [freq for freq in sorted_freqs['ppg']
                         if freq not in sorted_freqs['acc']][0]

            except:
                ind = sorted_inds['ppg'][0]
                est_f = freqs['ppg'][ind]

            est_hr = est_f * 60
            ref_hr = ref_hrs[idx.index(i)][0]
            errors.append(np.mean(np.abs(est_hr-ref_hr)))
            confidence.append(np.sum(mags['ppg'][(freqs['ppg'] >= est_f-30/60) & (
                        freqs['ppg'] <= est_f+30/60)]) / np.sum(mags['ppg']))
        return np.array(errors), np.array(confidence)

    Evaluate()
```

```
Out[2]: 16.162557800694298
```

```
In [ ]:
```